DD2434 Machine Learning, Advanced Course

Assignment 1

Jiang, Sifan
sifanj@kth.se

January 7, 2020

Contents

Listings

1 Knowing the Rules

Question 2.1.1: *It is mandatory to read the above text. Have you read it?*

Yes.

Question 2.1.2: *List all your collaborations concerning the problem formulations in this assignment.*

Hongsheng Chang.

Question 2.1.3: *Have you discussed solutions with anybody?*

No.

2 Dependencies in a Directed Graphical Model

Question 2.2.4: *In the graphical model of Figure 9, is $\mu_k \perp \tau_k$ (not conditioned by anything)?*

Yes.

Question 2.2.5: *In the graphical model of Figure 9, is $\mu_k \perp \tau_k \mid X^1, \cdots, X^N$?*

No.

Question 2.2.6: *In the graphical model of Figure 12, is $\mu \perp \beta'$ (not conditioned by anything)?*

Yes.

Question 2.2.7: *In the graphical model of Figure 12, is $\mu \perp \beta' \mid X^1, \cdots, X^N$?*

No.

Question 2.2.8: *In the graphical model of Figure 12, is $X^n \perp S^n$ (not conditioned by anything)?*

No.

Question 2.2.9: *In the graphical model of Figure 12, is $X^n \perp S^n \mid \mu_k, \tau_k$?*

No.

3 Likelihood of a tree GM only for E level

4 Simple VI

Question 2.4.12: *Implement the VI algorithm for the variational distribution in Equation (10.24) in Bishop.*

From *Bishop* [1], the variational distribution is given by

$$\begin{aligned} q(\mu, \tau) =& q_\mu(\mu) q_\tau(\tau) \\ q_\mu(\mu) =& \mathcal{N}(\mu \mid \mu_N, \lambda_N^{-1}) \\ q_\tau(\tau) =& \text{Gam}(\tau \mid a_N, b_N), \end{aligned}$$

where

$$\begin{aligned} \mu_N =& \frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N} \\ \lambda_N =& (\lambda_0 + N)\mathbb{E}[\tau] \\ a_N =& a_0 + \frac{N}{2} \\ b_N =& b_0 + \frac{1}{2}\mathbb{E}_\mu \left[\sum_{n=1}^{N} (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2 \right] \\ =& b_0 - \left(\sum_{n=1}^{N} x_n + \lambda_0 \mu_0 \right) \mathbb{E}_\mu[\mu] + \frac{1}{2} \left(\sum_{n=1}^{N} x_n^2 + \lambda_0 \mu_0^2 + (\lambda_0 + N)\mathbb{E}_\mu[\mu^2] \right). \end{aligned}$$

Where

$$\begin{aligned} \mathbb{E}_\mu[\mu] =& \mu_N \\ \mathbb{E}_\mu[\mu^2] =& \frac{1}{\lambda_N} + \mu_N^2 \\ \mathbb{E}_\tau[\tau] =& \frac{a_N}{b_N}. \end{aligned}$$

Due to non-informative condition, we set $\mu_0 = \lambda_0 = a_0 = b_0 = 0$. Then the VI algorithm would be an iterative algorithm that start with a set of initial values of μ_N, λ_N, a_N, and b_N. Then the values of μ_N, λ_N, a_N, and b_N should be updated to obtain a converging result of approximated posterior distribution. The algorithm is shown in the following code:

```
from math import exp, pi, sqrt
from scipy.special import gamma

import math
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(1111)
MAX_ITER = 100
THRES = 0.001

def generateData(N, mu=0.0, sigma=1.0):
    D = np.random.normal(mu, sigma, N)
    return(D)
```

```
def _plot(mu, tau, p, q, color):
    m, t = np.meshgrid(mu, tau)
    plt.figure()
    plt.contour(m, t, p)
    plt.contour(m, t, q, colors=color)
    plt.xlabel('$\\mu$')
    plt.ylabel('$\\tau$')
    plt.axis("equal")

# True posterior distribution
def _p(muT, lambdaT, aT, bT, mu, tau):
    p = (bT**aT)*sqrt(lambdaT) / (gamma(aT)*sqrt(2*pi)) * tau**(aT-0.5) \
        * np.exp(-bT*tau) * np.exp(-0.5*lambdaT*np.dot(tau,((mu-muT)**2).T))
    return p

# Approximated posterior distribution
def _q(muN, lambdaN, aN, bN, mu, tau):
    q_mu = sqrt(lambdaN/(2*pi)) * np.exp(-0.5 * np.dot(lambdaN, np.transpose((mu-muN
    )**2)))
    q_tau = (1.0/gamma(aN)) * bN**aN * tau**(aN-1) * np.exp(-bN*tau)
    q = q_tau * q_mu
    return q

# Update parameter
def _update(D, N, mu0, lambda0, a0, b0, muN, lambdaN, aN, bN):
    E_mu = muN
    E_mu2 = 1.0 / lambdaN + muN**2
    E_tau = aN / bN
    lambdaN = (lambda0 + N) * E_tau
    bN = b0 - (sum(D) + lambda0*mu0)*E_mu \
        + 0.5*(sum(D**2) + lambda0*mu0**2 + (lambda0+N)*E_mu2)
    return lambdaN, bN

def simpleVI():
    N = 10
    D = generateData(N)
    x_bar = D.mean()
    mu  = np.linspace(-2, 2, 100)
    tau = np.linspace( 0, 4, 100)
    a0 = 0
    b0 = 0
    mu0 = 0
    lambda0 = 0

    muT = (lambda0 * mu0 + N * x_bar) / (lambda0 + N)
    lambdaT = lambda0 + N
    aT = a0 + N/2
    bT = b0 + 0.5*sum((D-x_bar)**2) + (lambda0*N*(x_bar-mu0)**2)/(2*(lambda0+N))

    muN = (lambda0 * mu0 + N * x_bar) / (lambda0 + N)
    lambdaN = 0.1
    aN = a0 + (N + 1) / 2
    bN = 0.1
    lambdaOld = lambdaN
    bOld = bN

    p = _p(muT, lambdaT, aT, bT, mu[:,None], tau[:,None])

    for iter in range(MAX_ITER):
```

```
        lambdaN, bN = _update(D, N, mu0, lambda0, a0, b0, muN, lambdaN, aN, bN)
        q = _q(muN, lambdaN, aN, bN, mu[:,None], tau[:,None])

        if (abs(lambdaN - lambdaOld) < THRES) and (abs(bN - bOld) < THRES):
            _plot(mu, tau, p, q, 'r')
            plt.savefig("2_4_2_"+str(iter+1))
            break
        else:
            _plot(mu, tau, p, q, 'b')
            plt.savefig("2_4_2_"+str(iter+1))
        lambdaOld = lambdaN
        bOld = bN

    print(muN, lambdaN, aN, bN)

if __name__ == "__main__":
    simpleVI()
```

Listing 1: Simple VI.

Question 2.4.13: *What is the exact posterior?*

Since gamma distribution is a conjugate prior of the normal distribution, the joint distribution of the Normal-Gamma distribution is

$$p(\mu,\tau \mid \mu_0,\lambda_0,a_0,b_0) = \frac{b_0^{a_0}\sqrt{\lambda_0}}{\Gamma(a_0)\sqrt{2\pi}}\tau^{a_0-\frac{1}{2}}\exp\left[-b_0\tau\right]\exp\left[-\frac{\lambda\tau(\mu-\mu_0)^2}{2}\right]$$

$$p(\mu,\tau) \propto \tau^{a_0-\frac{1}{2}}\exp\left[-b_0\tau\right]\exp\left[-\frac{\lambda\tau(\mu-\mu_0)^2}{2}\right]$$

According to Bayesian theorem, we have

$$p(\mu,\tau \mid \mathcal{D}) \propto p(\mathcal{D} \mid \mu,\tau)p(\mu,\tau),$$

where $p(\mathcal{D} \mid \mu,\tau)$ is the likelihood of the data points which have such relationship:

$$\begin{aligned}
p(\mathcal{D} \mid \mu,\tau) &= \prod_{i=1}^{N} p(x_i \mid \mu,\tau) \\
&\propto \prod_{i=1}^{N} \tau^{1/2}\exp\left[-\frac{\tau}{2}(x_i-\mu)^2\right] \\
&\propto \tau^{N/2}\exp\left[-\frac{\tau}{2}\sum_{i=1}^{N}(x_i-\mu)^2\right] \\
&\propto \tau^{N/2}\exp\left[-\frac{\tau}{2}\sum_{i=1}^{N}(x_i-\bar{x}+\bar{x}-\mu)^2\right] \\
&\propto \tau^{N/2}\exp\left[-\frac{\tau}{2}\sum_{i=1}^{N}\left((x_i-\bar{x})^2+(\bar{x}-\mu)^2\right)\right] \\
&\propto \tau^{N/2}\exp\left[-\frac{\tau}{2}\left(Ns+N(\bar{x}-\mu)^2\right)\right],
\end{aligned}$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ and $s = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$. Thus the posterior can be written into [2]

$$\begin{aligned}
p(\mu, \tau \mid \mathcal{D}) \propto & p(\mathcal{D} \mid \mu, \tau) p(\mu, \tau) \\
\propto & \tau^{N/2} \exp\left[-\frac{\tau}{2}\left(Ns + N(\bar{x} - \mu)^2\right)\right] \tau^{a_0 - \frac{1}{2}} \exp\left[-b_0 \tau\right] \exp\left[-\frac{\lambda\tau(\mu - \mu_0)^2}{2}\right] \\
\propto & \tau^{\frac{N}{2} + a_0 - \frac{1}{2}} \exp\left[-\tau\left(\frac{1}{2}Ns + b_0\right)\right] \exp\left[-\frac{\tau}{2}\left(\lambda_0(\mu - \mu_0)^2 + N(\bar{x} - \mu)^2\right)\right].
\end{aligned}$$

The exponent (except the $-\frac{\tau}{2}$) in the last exponential term can be simplified into

$$\begin{aligned}
\lambda_0(\mu - \mu_0)^2 + N(\bar{x} - \mu)^2 = & \lambda_0\mu^2 - 2\lambda_0\mu\mu_0 + \lambda_0\mu_0^2 + N\mu^2 - 2N\bar{x}\mu + N\bar{x}^2 \\
= & (\lambda_0 + N)\mu^2 - 2(\lambda_0\mu_0 + N\bar{x})\mu + \lambda_0\mu_0^2 + N\bar{x}^2 \\
= & (\lambda_0 + N)\left(\mu^2 - 2\frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N}\mu\right) + \lambda_0\mu_0^2 + N\bar{x}^2 \\
= & (\lambda_0 + N)\left(\mu - \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N}\right)^2 + \lambda_0\mu_0^2 + N\bar{x}^2 - \frac{(\lambda_0\mu_0 + N\bar{x})^2}{\lambda_0 + N} \\
= & (\lambda_0 + N)\left(\mu - \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N}\right)^2 + \frac{\lambda_0 N(\bar{x} - \mu_0)^2}{\lambda_0 + N}.
\end{aligned}$$

Insert the above equation into the posterior distribution, we have

$$\begin{aligned}
p(\mu, \tau \mid \mathcal{D}) \propto & \tau^{\frac{N}{2} + a_0 - \frac{1}{2}} \exp\left[-\tau\left(\frac{1}{2}Ns + b_0\right)\right] \exp\left[-\frac{\tau}{2}\left((\lambda_0 + N)\left(\mu - \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N}\right)^2 + \frac{\lambda_0 N(\bar{x} - \mu_0)^2}{\lambda_0 + N}\right)\right] \\
\propto & \tau^{\frac{N}{2} + a_0 - \frac{1}{2}} \exp\left[-\tau\left(\frac{1}{2}Ns + b_0 + \frac{\lambda_0 N(\bar{x} - \mu_0)^2}{2(\lambda_0 + N)}\right)\right] \exp\left[-\frac{\tau}{2}(\lambda_0 + N)\left(\mu - \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N}\right)^2\right],
\end{aligned}$$

which is in the form of a Normal-Gamma distribution, thus the exact posterior is

$$\begin{aligned}
p(\mu, \tau \mid \mathcal{D}) = & \text{NormalGamma}(\mu', \lambda', a', b') \\
\mu' = & \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N} \\
\lambda' = & \lambda_0 + N \\
a' = & a_0 + \frac{N}{2} \\
b' = & b_0 + \frac{1}{2}\left(Ns + \frac{\lambda_0 N(\bar{x} - \mu_0)^2}{\lambda_0 + N}\right).
\end{aligned}$$

Question 2.4.14: *Compare the inferred variational distribution with the exact posterior. Run the inference on data points drawn from iid Gaussians. Do this for three interesting cases and visualize the results. Describe the differences.*

1. In this case, the number of data set is $N = 10$ and the initial values are $b_N = 5$ and $\lambda_N = 5$. The result is shown in fig 1, where the true posterior distribution is shown by the colorful contour lines, while the approximated distribution given by the VI algorithm is shown in blue contour lines and red contour lines when reached convergence. The convergence criteria is that both of b_N and λ_N are converge to a given threshold.

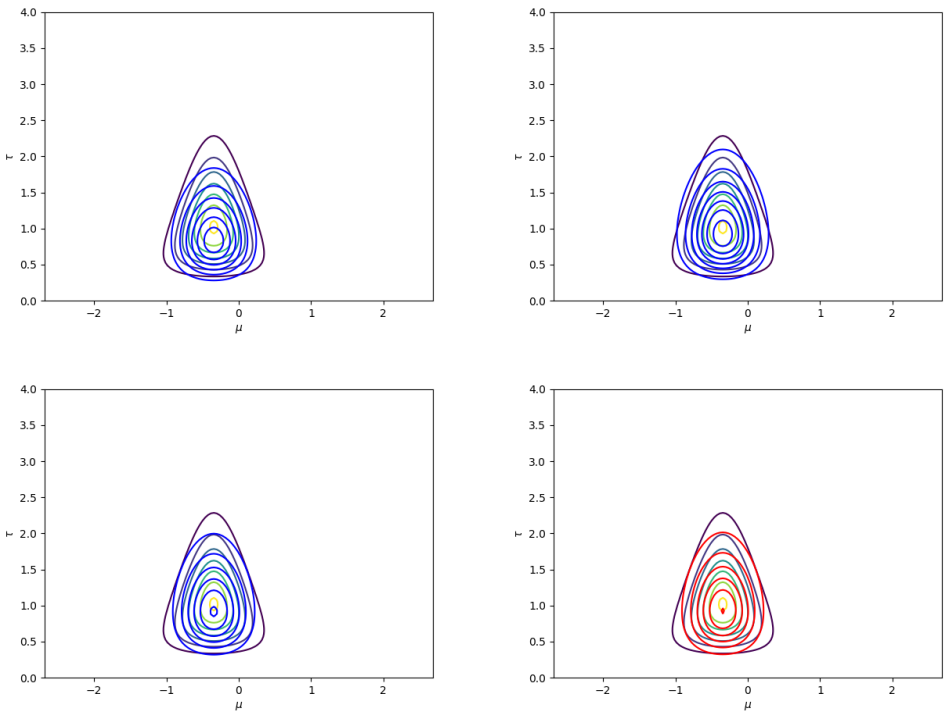


Figure 1: Simple VI with $\mu_0 = \lambda_0 = a_0 = b_0 = 0$, $b_N = 5$, and $\lambda_N = 5$ with 10 samples.

Although we are using zero for parameters in the prior distributions, so that $\mu_0 = \lambda_0 = a_0 = b_0 = 0$, which implies that we have no information about the distribution, the convergence is really quick with VI algorithm, which is 8 steps in my case.

And not only from this case, but also from the following cases, we can see that the mean of the μ is always correct, which is true since we get value μ_N at the beginning and don't update it during the iterations.

2. In this case, the number of data set is $N = 10$ and the initial values are $b_N = 0.1$ and $\lambda_N = 0.1$. The result is illustrated in fig 2.

   By setting the value of the initial values b_N and λ_N to smaller values which are much different to the true values in the true posterior distribution, the converging procedure is slower and requires more iterations, in my case 12 steps, to reach the convergence.

3. In this case, the number of data set is $N = 100$ and the initial values are $b_N = 5$ and $\lambda_N = 5$. The result is illustrated in fig 3.

   In this case, we use 100 data sets generated from a normal distribution. The true posterior distribution for μ and τ looks more like a circular shape. And since we have more data sets, we would be confident to the result and the convergence is quick at the same time, which is 8 steps.

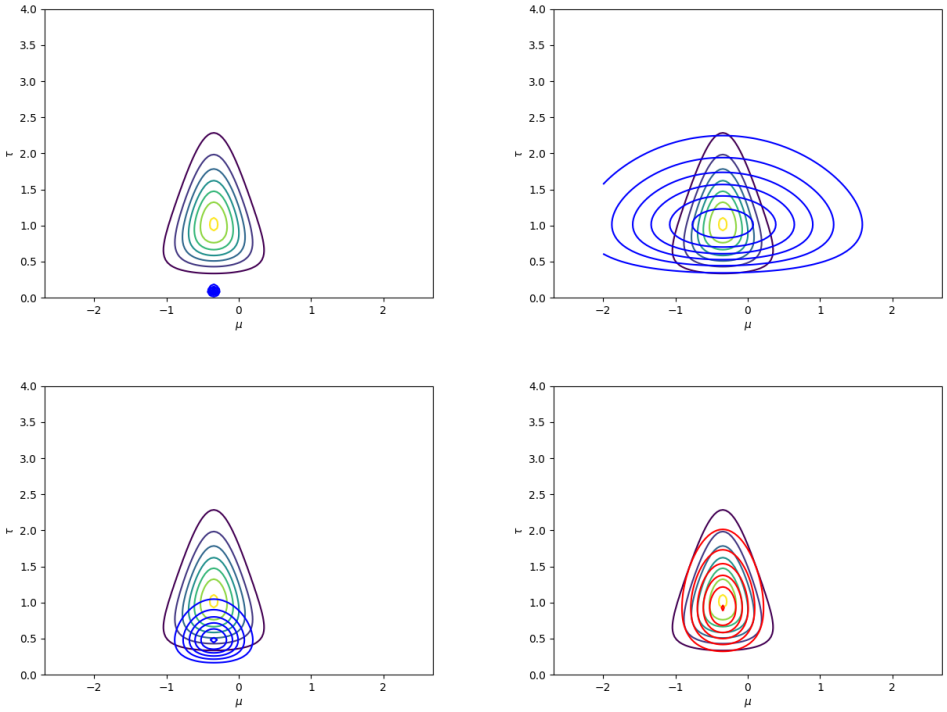


Figure 2: Simple VI with $\mu_0 = \lambda_0 = a_0 = b_0 = 0$, $b_N = 0.1$, and $\lambda_N = 0.1$ with 10 samples.

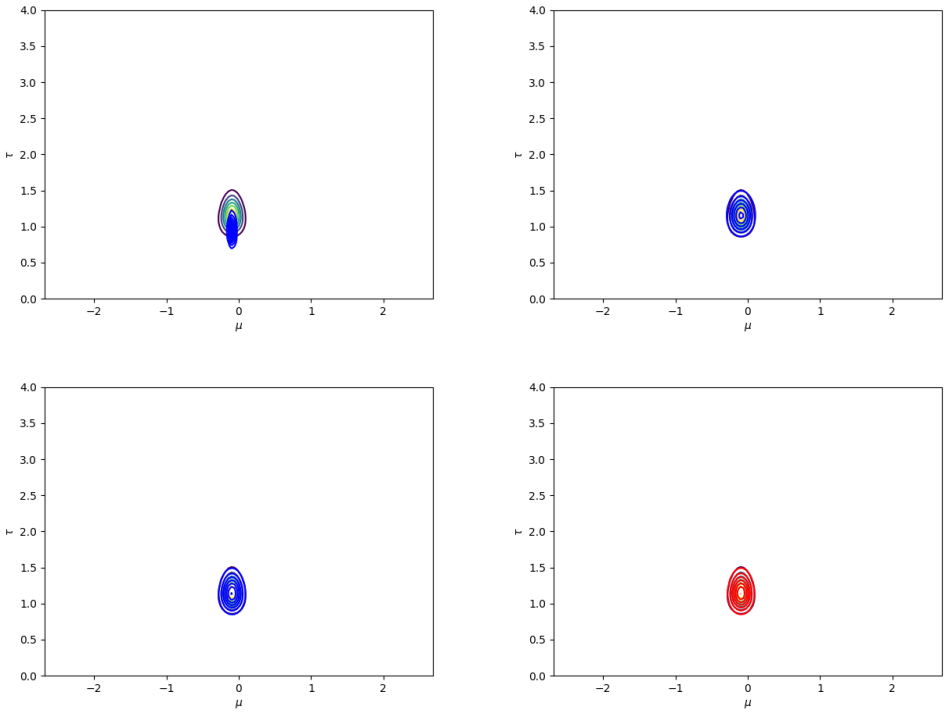


Figure 3: Simple VI with $\mu_0 = \lambda_0 = a_0 = b_0 = 0$, $b_N = 5$, and $\lambda_N = 5$ with 100 samples.

5 Mixture of trees with observable variables

Question 2.5.15: *Implement this EM algorithm.*

For each n, k the responsibilities is

$$\begin{aligned} r_{n,k} &= \frac{\pi_k p(x^n \mid T_k, \Theta_k)}{p(x^n)} \\ &= \frac{\pi_k p(x^n \mid T_k, \Theta_k)}{\sum_{k=1}^{K} \pi_k p(x^n \mid T_k, \Theta_k)}. \end{aligned}$$

The algorithm is implemented below, where the function in *Kruskal_v1.py* is slightly modified so that the result of the maximum spanning tree can be returned:

```
from Kruskal_v1 import Graph
from Tree import TreeMixture, Tree, Node

import numpy as np
import sys

epsilon = sys.float_info.epsilon

def em_algorithm(seed_val, samples, num_clusters, max_num_iter=100):
    print("Running EM algorithm...")

    # Set threshold for convergence
    THRES = 1e-4

    # Set rounds for sieving
    num_sieving = 10

    # Get the dimension of the data
    num_samples = np.size(samples, 0)
    num_nodes = np.size(samples, 1)

    # Sieving
    np.random.seed(seed_val)
    seeds = np.random.randint(0, 100000000, num_sieving)
    last_loglikelihoods = []
    tms = []
    for seed in seeds:
        np.random.seed(seed)
        tm = TreeMixture(num_clusters=num_clusters, num_nodes=num_nodes)
        tm.simulate_pi(seed_val=seed)
        tm.simulate_trees(seed_val=seed)
        tm_loglikelihood, tm = em_helper(tm, samples, num_clusters, max_num_iter=10)
        last_loglikelihoods.append(tm_loglikelihood[-1])
        tms.append(tm)

    # Main procedure for EM algorithm
    print("=> Sieving finished")
    seed = seeds[last_loglikelihoods.index(max(last_loglikelihoods))]
    tm = TreeMixture(num_clusters=num_clusters, num_nodes=num_nodes)
    tm.simulate_pi(seed_val=seed)
    tm.simulate_trees(seed_val=seed)
    loglikelihood, tm = em_helper(tm, samples, num_clusters, max_num_iter=
    max_num_iter)
```

```
    print("=> EM finished")
    topology_list = []
    theta_list = []
    for t in tm.clusters:
        topology_list.append(t.get_topology_array())
        theta_list.append(t.get_theta_array())
    loglikelihood = np.array(loglikelihood)
    topology_list = np.array(topology_list)
    theta_list = np.array(theta_list)

    return loglikelihood, topology_list, theta_list, tm

def em_helper(tm, samples, num_clusters, max_num_iter=10):
    num_samples = np.size(samples, 0)
    num_nodes = np.size(samples, 1)

    loglikelihood = []
    for iter in range(max_num_iter):
        print("========== "+str(iter)+"-th iteration ==========")
        # Step 1: Compute the responsibilities
        r = np.ones((num_samples, num_clusters))

        for n, x in enumerate(samples):
            for k, t in enumerate(tm.clusters):
                r[n,k] *= tm.pi[k]
                visit_list = [t.root]
                while len(visit_list) is not 0:
                    cur_node = visit_list[0]
                    visit_list = visit_list[1:]
                    visit_list = visit_list + cur_node.descendants
                    if cur_node.ancestor is None:
                        r[n,k] *= cur_node.cat[x[int(cur_node.name)]]
                    else:
                        r[n,k] *= cur_node.cat[x[int(cur_node.ancestor.name)]][x[int(cur_node.name)]]

        r += epsilon
        marginal = np.reshape(np.sum(r, axis=1), (num_samples, 1))
        loglikelihood.append(np.sum(np.log(marginal)))
        marginal_expand = np.repeat(marginal, num_clusters, axis=1)
        r /= marginal_expand

        # Step 2: Update categorical distribution
        tm.pi = np.mean(r, axis=0)

        # Step 3: Construct directed graphs
        denom = np.sum(r, axis=0)
        q = np.zeros((num_nodes, num_nodes, 2, 2, num_clusters)) # (s, t, a, b, k)
        for s in range(num_nodes):
            for t in range(num_nodes):
                for a in range(2):
                    for b in range(2):
                        index = np.where((samples[:,(s,t)]==[a,b]).all(1))[0]
                        numer = np.sum(r[index], axis=0)
                        q[s, t, a, b] = numer / denom
        q += epsilon

```

```
        q_s = np.zeros((num_nodes, 2, num_clusters))
        for s in range(num_nodes):
            for a in range(2):
                index = np.where(samples[:, s]==a)
                numer = np.sum(r[index], axis=0)
                q_s[s,a] = numer / denom
        q_s += epsilon

        I = np.zeros((num_nodes, num_nodes, num_clusters)) # (s, t, k)
        for s in range(num_nodes):
            for t in range(num_nodes):
                for a in range(2):
                    for b in range(2):
                        I[s,t] += q[s,t,a,b] * np.log(q[s,t,a,b] / q_s[s,a] / q_s[t,
    b])

        clusters = []
        for k in range(num_clusters):
            g = Graph(num_nodes)
            for s in range(num_nodes):
                for t in range(s+1, num_nodes):
                    g.addEdge(s, t, I[s, t, k])

            # Step 4: Construct maximum spanning trees
            edges = np.array(g.maximum_spanning_tree())[:,0:2]
            topology_array = np.zeros(num_nodes)
            topology_array[0] = np.nan
            visit_list = [0]
            while len(visit_list) != 0:
                cur_node = visit_list[0]
                index = np.where(edges==cur_node)
                index = np.transpose(np.stack(index))
                visit_list = visit_list[1:]
                for id in index:
                    child = edges[id[0], 1-id[1]]
                    topology_array[int(child)] = cur_node
                    visit_list.append(int(child))
                if np.size(index) is not 0:
                    edges = np.delete(edges, index[:,0], axis=0)

            tree = Tree()
            tree.load_tree_from_direct_arrays(topology_array)
            tree.k = 2
            tree.alpha = [1.0] * 2

            # Step 5: Update CPDs
            visit_list = [tree.root]
            while len(visit_list) != 0:
                cur_node = visit_list[0]
                visit_list = visit_list[1:]
                visit_list = visit_list + cur_node.descendants
                if cur_node.ancestor is None:
                    cur_node.cat = q_s[int(cur_node.name),:,k].tolist()
                else:
                    cat = q[int(cur_node.ancestor.name),int(cur_node.name),:,:,k]
                    cur_node.cat = [cat[0].tolist(), cat[1].tolist()]

            clusters.append(tree)
        tm.clusters =clusters
```

```
159     return loglikelihood , tm
```

Listing 2: EM algorithm for mixture of trees with observable variables.

Question 2.5.16: *Apply your algorithm to the provided data and show how well you reconstruct the mixtures. First, compare the real and inferred trees with the unweighted Robinson-Foulds (aka symmetric difference) metric. Do the trees have similar structure (don't worry if the inferred trees don't match with the real trees)? Then, compare the likelihoods of real and inferred mixtures. Finally, simulate more data and analyze the results (try to find some interesting and more challenging cases).*

- ***q_2_5_tm_10node_20sample_4clusters*:** The result of the EM algorithm is illustrated in fig 4.

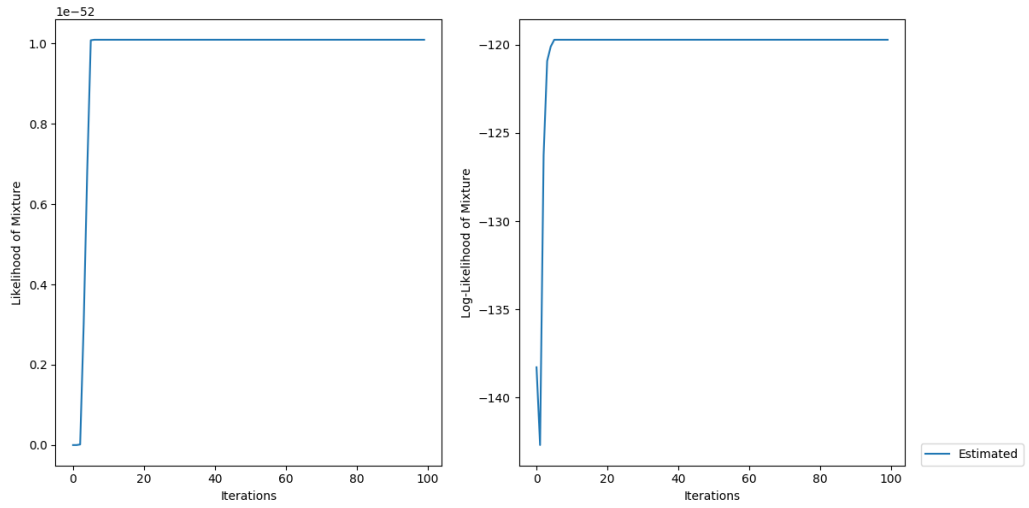


Figure 4: The likelihood and log-likelihood at each iteration of the EM algorithm for data *q_2_5_tm_10node_20sample_4clusters*.

The comparison of the real and inferred trees with unweighted RF metric is show in tab 1, from which we can see that the trees don't have similar structures which can caused by small number of samples.

Table 1: RF distance between the true tree mixture and the result obtained from the EM algorithm for data *q_2_5_tm_10node_20sample_4clusters*, where the rows represents the obtained tree mixture and the columns represents the true tree mixture.

	0	1	2	3
0	9	8	10	10
1	9	12	10	10
2	8	7	9	9
3	8	11	11	9

The log-likelihood of the result tree mixture obtained by the EM algorithm is -119.724756, while the log-likelihood of the ground truth is -113.143100.

Also, in the next question, 1000 data points would be sampled from the tree mixture to see what would happen with the result of the EM algorithm.

- ***q_2_5_tm_10node_50sample_4clusters*:** The result of the EM algorithm is illustrated in fig 5.

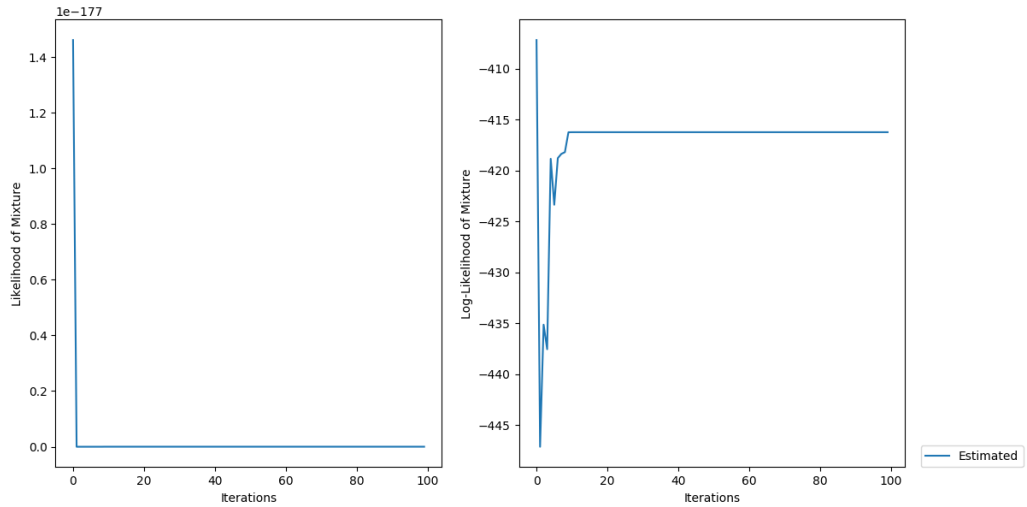


Figure 5: The likelihood and log-likelihood at each iteration of the EM algorithm for data *q_2_5_tm_10node_50sample_4clusters*.

The comparison of the real and inferred trees with unweighted RF metric is show in tab 2. Still the trees don't have similar structures but the distance looks more average than the one with only 20 samples.

Table 2: RF distance between the true tree mixture and the result obtained from the EM algorithm for data *q_2_5_tm_10node_50sample_4clusters*, where the rows represents the obtained tree mixture and the columns represents the true tree mixture.

	0	1	2	3
0	10	9	11	9
1	11	8	10	10
2	7	10	10	8
3	10	9	9	11

The log-likelihood of the result tree mixture obtained by the EM algorithm is -416.226204, while the log-likelihood of the ground truth is -280.856624.

- ***q_2_5_tm_20node_20sample_4clusters*:** The result of the EM algorithm is illustrated in fig 6.

The comparison of the real and inferred trees with unweighted RF metric is show in tab 3, from which we can see that the trees don't have similar structures which can caused by small

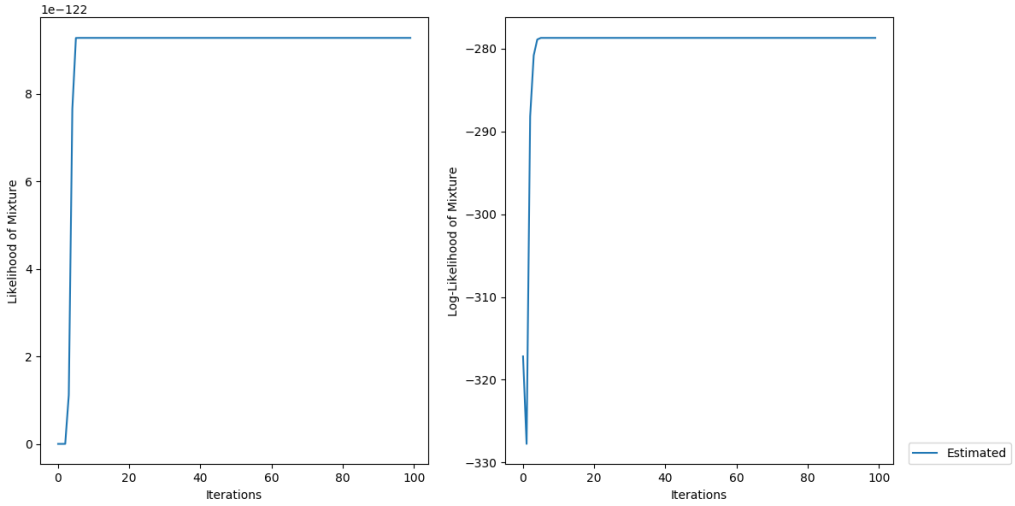


Figure 6: The likelihood and log-likelihood at each iteration of the EM algorithm for data *q_2_5_tm_20node_20sample_4clusters*.

number of samples and the result is even worse than the one with 10 nodes due to the number of nodes is bigger, thus making the inference much harder and making it more difficult to have similar structures.

Table 3: RF distance between the true tree mixture and the result obtained from the EM algorithm for data *q_2_5_tm_20node_20sample_4clusters*, where the rows represents the obtained tree mixture and the columns represents the true tree mixture.

	0	1	2	3
0	19	19	19	20
1	19	19	19	22
2	19	15	17	18
3	18	20	24	21

The log-likelihood of the result tree mixture obtained by the EM algorithm is -278.687237, while the log-likelihood of the ground truth is -217.005755.

Question 2.5.17: *Simulate new tree mixtures with different number of nodes, samples and clusters. Try to find some interesting cases. Analyze your results as in the previous question.*

By simulating a tree mixture with only 3 nodes and 10 clusters and sample 1000 data points from the tree mixture, the result is shown in fig 7.

The comparison of the real and inferred trees with unweighted RF metric is show in tab 4, from which we can see that the trees have similar structures because the number of nodes is only three which can be easily inferred.

The log-likelihood of the result tree mixture obtained by the EM algorithm is -1997.297679, while the log-likelihood of the ground truth is -2003.252791.

Another example simulating a tree mixture with 10 nodes and 10 clusters, and sample 50 data points from the tree mixture (notice that it is similar to *q_2_5_tm_10node_50sample_4clusters* while

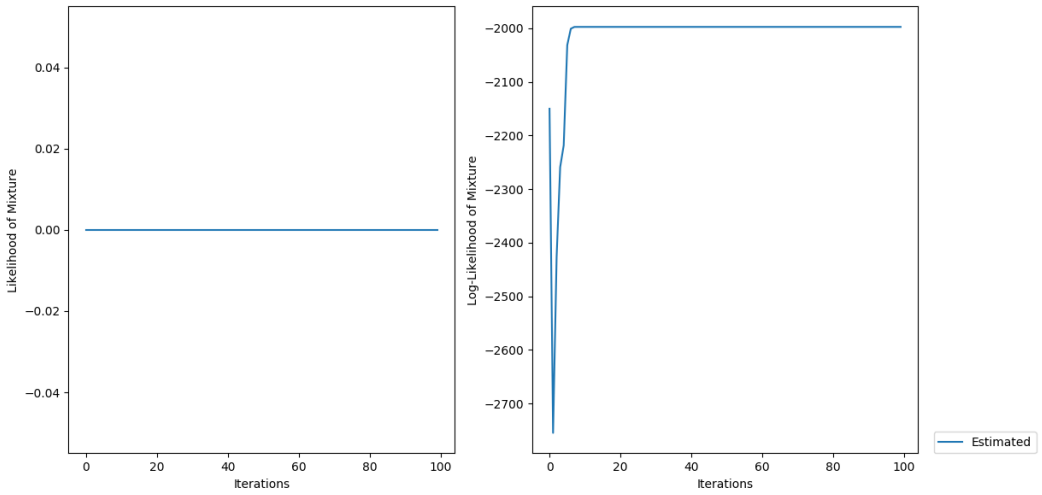


Figure 7: The likelihood and log-likelihood at each iteration of the EM algorithm for tree mixture with 3 nodes and 10 clusters.

Table 4: RF distance between the true tree mixture and the result obtained from the EM algorithm for tree mixture with 3 nodes and 10 clusters, where the rows represents the obtained tree mixture and the columns represents the true tree mixture.

	0	1	2	3	4	5	6	7	8	9
0	2	2	0	0	2	0	2	2	0	2
1	2	2	0	0	2	0	2	2	0	2
2	2	2	0	0	2	0	2	2	0	2
3	0	0	2	2	0	2	0	0	2	0
4	2	2	0	0	2	0	2	2	0	2
5	0	0	2	2	0	2	0	0	2	0
6	2	2	0	0	2	0	2	2	0	2
7	0	0	2	2	0	2	0	0	2	0
8	0	0	2	2	0	2	0	0	2	0
9	2	2	0	0	2	0	2	2	0	2

the only difference is the number of clusters), the result of the EM algorithm is illustrated in fig 8.

The comparison of the real and inferred trees with unweighted RF metric is show in tab 5, from which we can see that the trees have more similar structures when compared to the case of *q_2_5_tm_10node_50sample_4clusters*.

The log-likelihood of the result tree mixture obtained by the EM algorithm is -359.954244, while the log-likelihood of the ground truth is -323.847884. Which are relatively similar when compare to the case of *q_2_5_tm_10node_50sample_4clusters*.

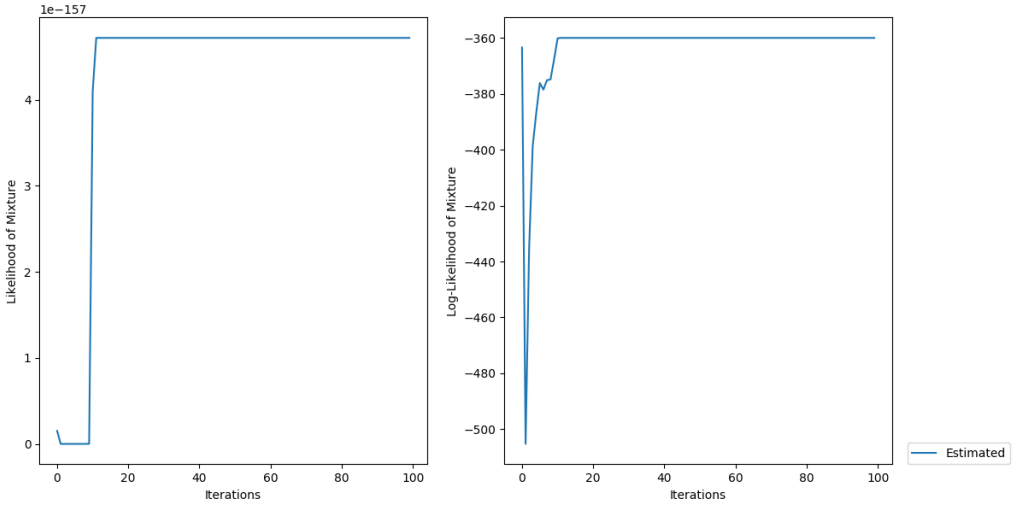


Figure 8: The likelihood and log-likelihood at each iteration of the EM algorithm for tree mixture with 10 nodes and 10 clusters.

Table 5: RF distance between the true tree mixture and the result obtained from the EM algorithm for tree mixture with 10 nodes and 10 clusters, where the rows represents the obtained tree mixture and the columns represents the true tree mixture.

	0	1	2	3	4	5	6	7	8	9
0	9	10	8	7	7	9	5	11	9	6
1	10	9	7	12	10	6	8	10	8	9
2	11	8	8	11	9	7	7	11	9	8
3	10	9	7	10	8	10	6	12	8	7
4	9	8	8	9	9	7	9	9	9	8
5	10	9	9	10	12	10	10	10	10	11
6	9	10	8	7	9	9	7	11	9	8
7	10	7	7	8	8	8	8	10	8	7
8	9	8	10	9	11	11	9	9	11	10
9	11	8	8	9	9	7	9	11	9	8

6 Super epicentra - EM

Question 2.6.18: *Derive an EM algorithm for the model.*

The conditional distribution of X^n, S^n given a particular value for Z^n is

$$\begin{aligned} p(X^n, S^n | Z_k^n = 1) =& \mathcal{N}(X^n | \mu_k, \tau_k) \text{Poisson}(S^n | \lambda_k) \\ =& \frac{\sqrt{\tau_{k,1}\tau_{k,2}}}{2\pi} \exp\left[-\frac{\tau_{k,1}}{2}(X_1^n - \mu_{k,1})^2 - \frac{\tau_{k,2}}{2}(X_2^n - \mu_{k,2})^2\right] \frac{\lambda_k^{S^n}}{S^n!} \exp\left[-\lambda_k\right] \\ =& \frac{\sqrt{\tau_{k,1}\tau_{k,2}}}{2\pi} \frac{\lambda_k^{S^n}}{S^n!} \exp\left[-\frac{\tau_{k,1}}{2}(X_1^n - \mu_{k,1})^2 - \frac{\tau_{k,2}}{2}(X_2^n - \mu_{k,2})^2 - \lambda_k\right] \end{aligned}$$

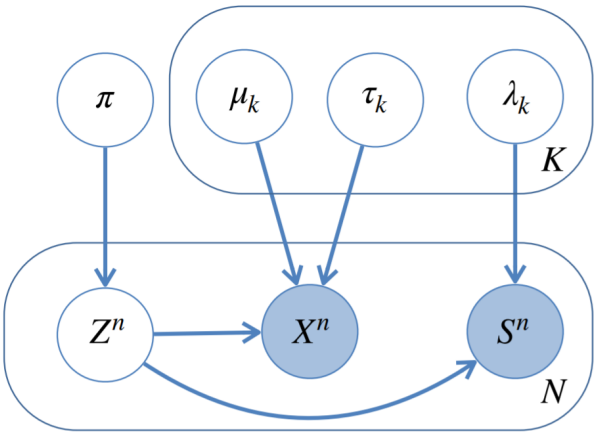


Figure 9: Mixture of components modeling location and strengths of earthquakes associated with a super-epicentra. In the figure, $\mu_k = (\mu_{k,1}, \mu_{k,2})$ and $\tau_k = (\tau_{k,1}, \tau_{k,2})$.

which can also be written in the form

$$p(X^n, S^n|Z^n) = \prod_{k=1}^{K} p(X^n, S^n|Z_k^n = 1)^{Z_k^n}.$$

The joint distribution is given by

$$p(X^n, S^n, Z^n) = p(Z^n)p(X^n, S^n|Z^n),$$

and the marginal distribution of X^n, S^n is then obtained by summing the joint distribution over all possible states of Z^n to give

$$p(X^n, S^n) = \sum_{k=1}^{K} \pi_k p(X^n, S^n|Z_k^n = 1).$$

Another quantity that will play an important role is the conditional probability of Z^n given X^n, S^n. We shall use $\gamma(Z_k^n)$ to denote $p(Z_k^n = 1|X^n, S^n)$, whose value can be found using Bayes' theorem

$$\begin{aligned}\gamma(Z_k^n) \equiv p(Z_k^n = 1|X^n, S^n) &= \frac{p(Z_k^n = 1)p(X^n, S^n|Z_k^n = 1)}{\sum_{j=1}^{K} p(Z_j^n = 1)p(X^n, S^n|Z_j^n = 1)} \\ &= \frac{\pi_k p(X^n, S^n|Z_k^n = 1)}{\sum_{j=1}^{K} \pi_j p(X^n, S^n|Z_j^n = 1)}.\end{aligned}$$

The M-step in general EM algorithm is to evaluate $\boldsymbol{\theta}^{\text{new}}$ given by

$$\boldsymbol{\theta}^{\text{new}} = \arg_{\boldsymbol{\theta}} \max \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$$

where

$$\begin{aligned}\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \mathbf{S}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X}, \mathbf{S}, \mathbf{Z}|\boldsymbol{\theta}) \\ &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \mathbf{S}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{Z}|\boldsymbol{\theta}) p(\mathbf{X}, \mathbf{S}|\mathbf{Z}, \boldsymbol{\theta})\end{aligned}$$

Setting the derivatives of $p(\mathbf{Z}|\mathbf{X},\mathbf{S},\boldsymbol{\theta}^{\text{old}})\ln p(\mathbf{X},\mathbf{S},\mathbf{Z}|\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ to zero, we have the updated parameters given by

$$\begin{aligned}
\mu_k^{\text{new}} =& \frac{1}{N_k}\sum_{n=1}^{N}\gamma(Z_k^n)X^n \\
\tau_{k,1}^{\text{new}} =& \frac{N_k}{\sum_{n=1}^{N}\gamma(Z_k^n)(X_1^n-\mu_{k,1}^{\text{new}})^2} \\
\tau_{k,2}^{\text{new}} =& \frac{N_k}{\sum_{n=1}^{N}\gamma(Z_k^n)(X_2^n-\mu_{k,2}^{\text{new}})^2} \\
\lambda_k^{\text{new}} =& \frac{1}{N_k}\sum_{n=1}^{N}\gamma(Z_k^n)S^n \\
\pi_k^{\text{new}} =& \frac{N_k}{N}.
\end{aligned}$$

EM algorithm:

1. Initialize the means μ_k, precision τ_k, Poisson parameter λ_k and mixing coefficients π_k, and evaluate the initial value of the log likelihood.

2. **E Step.** Evaluate the responsibilities using the current parameter values

$$\gamma(Z_k^n) = \frac{\pi_k p(X^n,S^n|Z_k^n=1)}{\sum_{j=1}^{K}\pi_j p(X^n,S^n|Z_j^n=1)}.$$

3. **M Step.** Re-estimate the parameters using the current responsibilities

$$\begin{aligned}
\mu_k^{\text{new}} =& \frac{1}{N_k}\sum_{n=1}^{N}\gamma(Z_k^n)X^n \\
\tau_{k,1}^{\text{new}} =& \frac{N_k}{\sum_{n=1}^{N}\gamma(Z_k^n)(X_1^n-\mu_{k,1}^{\text{new}})^2} \\
\tau_{k,2}^{\text{new}} =& \frac{N_k}{\sum_{n=1}^{N}\gamma(Z_k^n)(X_2^n-\mu_{k,2}^{\text{new}})^2} \\
\lambda_k^{\text{new}} =& \frac{1}{N_k}\sum_{n=1}^{N}\gamma(Z_k^n)S^n \\
\pi_k^{\text{new}} =& \frac{N_k}{N}
\end{aligned}$$

   where

$$N_k = \sum_{n=1}^{N}\gamma(Z_k^n).$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X},\mathbf{S}|\boldsymbol{\mu},\boldsymbol{\tau},\boldsymbol{\lambda},\boldsymbol{\pi}) = \sum_{n=1}^{N}\ln\left\{\sum_{k=1}^{K}\pi_k\mathcal{N}(X^n|\mu_k,\tau_k)\text{Poisson}(S^n|\lambda_k)\right\}$$

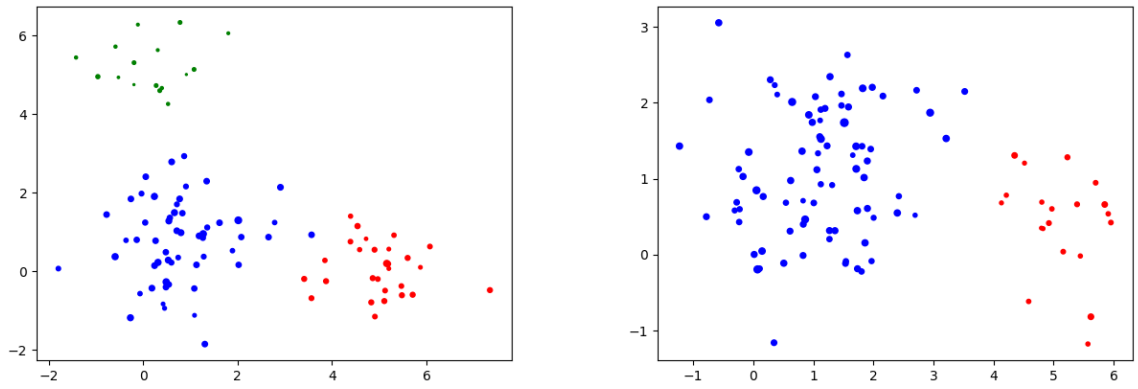
   and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

Question 2.6.19: *Implement your EM algorithm.*

```
def _do_estep(self, X, S):
    """
    E-step
    """
    r = np.zeros((self.n_row, self.n_components)) # r[n,k]
    for n in range(self.n_row):
        for k in range(self.n_components):
            r[n,k] = self.weights[k] \
                   * multivariate_normal(self.means[k], self.covs[k]).pdf(X[n]) \
                   * poisson(self.rates[k]).pmf(S[n])

    r += epsilon
    marginal = np.sum(r, axis=1).reshape((self.n_row, 1))
    marginal = np.repeat(marginal, self.n_components, axis=1)
    r /= marginal

    self.r = r
    return self

def _do_mstep(self, X, S):
    """M-step, update parameters"""
    N = np.sum(self.r, axis=0) # N[k] = Nk
    N_expand = np.repeat(N.reshape((self.n_components, 1)), self.n_col, axis=1)
    self.means = np.dot(np.transpose(self.r), X) / N_expand

    for k in range(self.n_components):
        mean_expand = np.repeat(np.atleast_2d(self.means[k]), self.n_row, axis=0)
        r_expand = np.repeat(self.r[:,k].reshape((self.n_row, 1)), self.n_col, axis
    =1)
        variance = np.sum(r_expand * (X - mean_expand)**2, axis=0) / N[k]
        self.covs[k] = np.diag(variance)

    self.rates = np.dot(S, self.r) / N

    self.weights = N / self.n_row

    return self

def _compute_log_likelihood(self, X, S):
    """compute the log likelihood of the current parameter"""
    log_likelihood = 0
    for n in range(self.n_col):
        likelihood = 1
        for k in range(self.n_components):
            likelihood *= self.weights[k] \
                        * multivariate_normal(self.means[k], self.covs[k]).pdf(X[n])
     \
                        * poisson(self.rates[k]).pmf(S[n])
        log_likelihood += np.log(likelihood)

    return log_likelihood
```

Listing 3: EM algorithm for super epicentra.

Question 2.6.20: *Apply it to the data provided separately, give an account of the success, and provide visualizations for a couple of examples.*



(a) Data set 1.

(b) Data set 2.

Figure 10: Ground truth for data sets

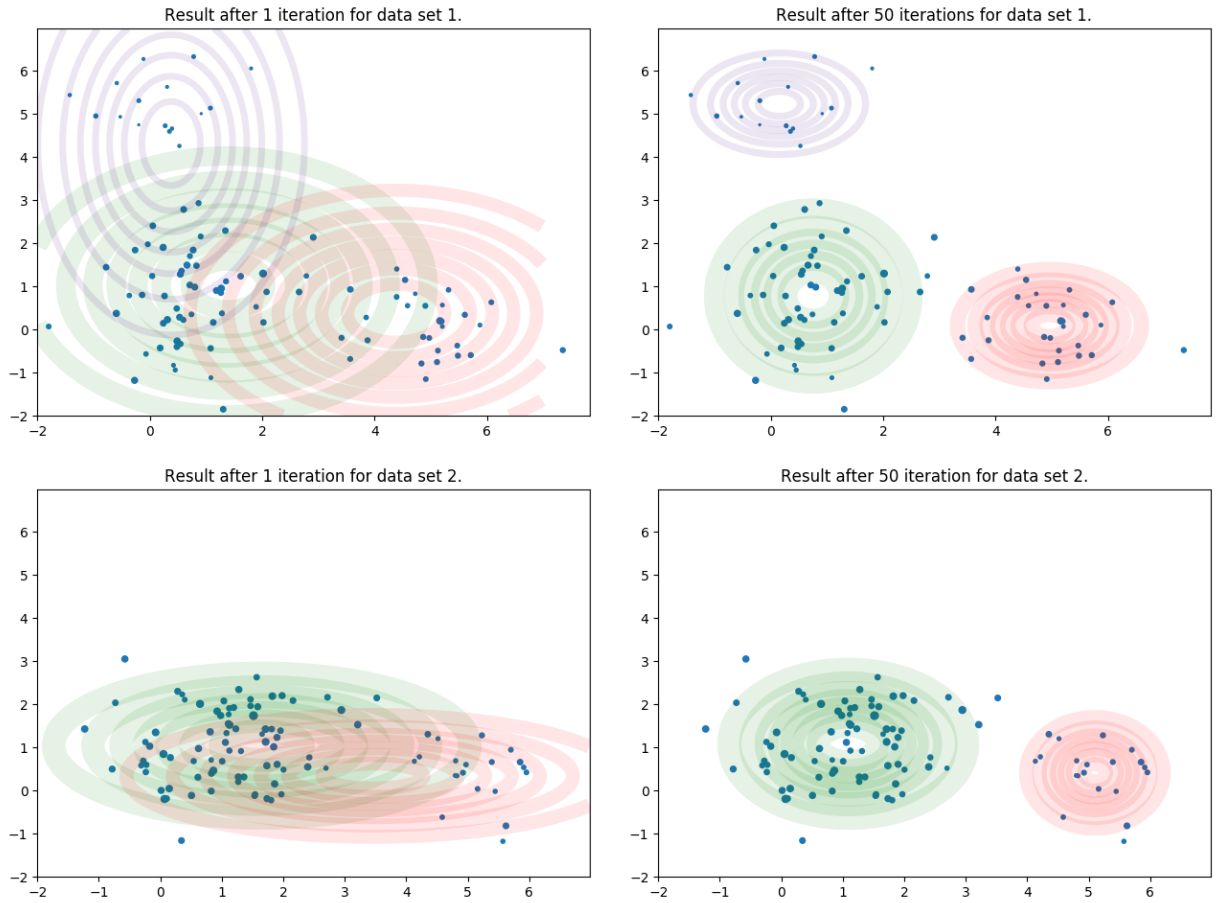


Figure 11: Result of the EM algorithm for different data sets.

The ground truth of the data is illustrated in fig 10 and the inference result from EM algorithm

is illustrated in fig 11. From those two figures, we can tell that the result of EM algorithm gives good inference of the means and covariances for each Gaussian model, and ratios for each Poisson model.

7 Super epicentra - VI

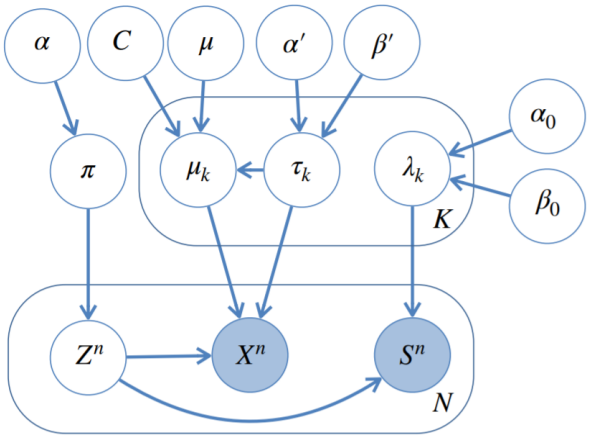


Figure 12: The K super epicentra model with priors.

Question 2.7.21: *Derive a VI algorithm that estimates the posterior distribution for this model.*

The VI algorithm is derived based on [1] and lecture notes.
Since $\pi \sim \text{Dir}(\alpha)$, we have

$$\begin{aligned} p(\pi|\alpha) =& p(\pi_1, \cdots, \pi_K|\alpha_1, \cdots, \alpha_K) \\ =& \frac{\Gamma\left(\sum_{k=1}^K \alpha_k\right)}{\prod_{k=1}^K \Gamma(\alpha_k)} \prod_{k=1}^K \pi_k^{\alpha_k - 1} \\ =& \frac{1}{B(\alpha)} \prod_{k=1}^K \pi_k^{\alpha_k - 1}. \end{aligned}$$

Also, for other latent variables, we have

$$\begin{aligned} p(\mu_k|\tau_k) =& \mathcal{N}(\mu_k|\mu, (C\tau_k)^{-1}) \\ p(\tau_k) =& \text{Gam}(\tau_k|\alpha', \beta') \\ p(\lambda_k) =& \text{Gam}(\lambda_k|\alpha_0, \beta_0). \end{aligned}$$

The joint distribution of all of the random variables is given by

$$p(\mathbf{X}, \mathbf{S}, \mathbf{Z}, \pi, \mu, \tau, \lambda) = p(\mathbf{X}|\mathbf{Z}, \mu, \tau)p(\mathbf{S}|\mathbf{Z}, \lambda)p(\mathbf{Z}|\pi)p(\pi)p(\mu|\tau)p(\tau)p(\lambda).$$

A variational distribution which factorizes between the latent variables and the parameters is given by

$$q(\mathbf{Z}, \pi, \mu, \tau, \lambda) = q(\mathbf{Z})q(\pi, \mu, \tau, \lambda).$$

The log of the optimized factor is then given by

$$\begin{aligned}\ln q^*(\mathbf{Z}) =& \mathbb{E}_{\pi,\mu,\tau,\lambda}[\ln p(\mathbf{X},\mathbf{S},\mathbf{Z},\pi,\mu,\tau,\lambda)] + \text{const} \\ =& \mathbb{E}_{\pi}[\ln p(\mathbf{Z}|\pi)] + \mathbb{E}_{\mu,\tau}[\ln p(\mathbf{X}|\mathbf{Z},\mu,\tau)] + \mathbb{E}_{\lambda}[\ln p(\mathbf{S}|\mathbf{Z},\lambda)] + \text{const} \\ =& \sum_{n=1}^{N}\sum_{k=1}^{K} Z_k^n \ln \rho_{nk} + \text{const}\end{aligned}$$

where

$$\begin{aligned}\ln \rho_{nk} =& \mathbb{E}\left[\ln \pi_k\right] + \frac{1}{2}\mathbb{E}\left[\ln|\tau_k|\right] - \ln(2\pi) \\ & - \frac{1}{2}\mathbb{E}_{\mu_k,\tau_k}\left[(X^n-\mu_k)^{\mathrm{T}}\tau_k(X^n-\mu_k)\right] \\ & + S^n\mathbb{E}[\ln\lambda_k] - \mathbb{E}[\lambda_k] - \ln\left(S^n!\right).\end{aligned}$$

Taking the exponential of both sides of the equation and normalize it, we obtain

$$q^*(\mathbf{Z}) = \prod_{n=1}^{N}\prod_{k=1}^{K} r_{nk}^{Z_k^n}$$

where

$$r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^{K}\rho_{nj}}.$$

For the discrete distribution $q^*(\mathbf{Z})$ we have

$$\mathbb{E}[Z_k^n] = r_{nk}$$

and define

$$N_k = \sum_{n=1}^{N} r_{nk}.$$

Then consider the factor $q(\pi,\mu,\tau,\lambda)$ in the variational posterior distribution, we have

$$\begin{aligned}\ln q^*(\pi,\mu,\tau,\lambda) =& \ln p(\pi) + \mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{Z}|\pi)] + \text{const} \\ & + \sum_{k=1}^{K}\ln p(\mu_k,\tau_k) + \sum_{k=1}^{K}\sum_{n=1}^{N}\mathbb{E}[Z_k^n]\ln\mathcal{N}(X^n|\mu_k,\tau_k^{-1}) \\ & + \sum_{k=1}^{K}\ln p(\lambda_k) + \sum_{k=1}^{K}\sum_{n=1}^{N}\mathbb{E}[Z_k^n]\ln\text{Poisson}(S^n|\lambda_k).\end{aligned}$$

Since the right-hand side of this expression decomposes into a sum of terms involving only π, terms only involving μ and τ, and terms only involving λ, which implies that the variational posterior $q(\pi,\mu,\tau,\lambda)$ factorizes to give $q(\pi)q(\mu,\tau)q(\lambda)$ which gives

$$q(\pi,\mu,\tau,\lambda) = q(\pi)\prod_{k=1}^{K} q(\mu_k,\tau_k)q(\lambda_k).$$

- First, by identifying the terms that depend on π, we have

$$\ln q^*(\pi) = (\alpha - 1) \sum_{k=1}^{K} \ln \pi_k + \sum_{k=1}^{K} \sum_{n}^{N} r_{nk} \ln \pi_k + \text{const}$$

  and taking the exponential of both sides, we have

$$q^*(\pi) = \text{Dir}(\pi | \alpha^*)$$

  where α^* has components α_k^* given by

$$\alpha_k^* = \alpha_k + N_k.$$

- Similarly, by identifying the terms that depends on μ_k and τ_k, we have

$$\begin{aligned}
\ln q^*(\mu_k, \tau_k) = & \sum_{n=1}^{N} r_{nk} \left(\frac{1}{2} \ln \tau_k - \frac{1}{2} \tau_k \left(X^n - \mu_k \right)^2 \right) \\
& + \frac{1}{2} \ln \tau_k - \frac{1}{2} C (\mu_k - \mu)^2 \\
& + (\alpha' - 1) \ln \tau_k - \beta' \tau_k.
\end{aligned}$$

  So, we have

$$\begin{aligned}
q^*(\mu_k | \tau_k) = & \mathcal{N}(\mu_k | \mu^*, \tau^*) \\
\tau^* = & (C + N_k) \tau_k \\
\mu^* = & \frac{\left(\tau_k \sum_{n=1}^{N} r_{nk} X^n \right) + C \tau_k \mu}{\tau^*},
\end{aligned}$$

  and

$$\begin{aligned}
q^*(\tau_k) = & \, \text{Gam}(\tau_k | \alpha'^*, \beta'^*) \\
\alpha'^* = & \alpha' + N_k \\
\beta'^* = & \beta' + \frac{1}{2} C \mu^2 + \frac{1}{2} \sum_{n=1}^{N} r_{nk} X^{n2}.
\end{aligned}$$

- Finally, by identifying the terms that depends on λ_k, we have

$$\begin{aligned}
\ln q^*(\lambda_k) = & (\alpha_0 - 1) \ln \lambda_k - \beta_0 \lambda_k + \text{const} \\
& + \sum_{n=1}^{N} \left(r_{nk} \left(S^n \ln \lambda_k - \lambda_k - \ln \left(S^n ! \right) \right) \right).
\end{aligned}$$

  So, we have

$$\begin{aligned}
q^*(\lambda_k) = & \, \text{Gam}(\lambda_k | \alpha_0^*, \beta_0^*) \\
\alpha_0^* = & \alpha_0 + \sum_{n=1}^{N} r_{nk} S^n \\
\beta_0^* = & \beta + N_k.
\end{aligned}$$

8 Sampling from a tree GM

Question 2.8.22: *Derive these algorithms.*

1. **Bottom-up Dynamic Programming (DP) algorithm:**

   Define $\mathfrak{O}\{T\}$ such that the sum of the leaves of tree T is odd. Similarly, define $\mathfrak{E}\{T\}$ such that the sum of the leaves of tree T is even. Also, define $\mathcal{L}(T)$ to be the left subtree of the root of the tree T, and define $\mathcal{R}(T)$ to be the right subtree of the root of the tree T, which both can be empty tree. Then for an arbitrary tree, we have

$$\begin{aligned} p(\mathfrak{O}\{T\}|T,\Theta) =& p(\mathfrak{O}\{\mathcal{L}(T)\}|\mathcal{L}(T),\Theta)p(\mathfrak{E}\{\mathcal{R}(T)\}|\mathcal{R}(T),\Theta) \\ &+p(\mathfrak{E}\{\mathcal{L}(T)\}|\mathcal{L}(T),\Theta)p(\mathfrak{O}\{\mathcal{R}(T)\}|\mathcal{R}(T),\Theta) \\ p(\mathfrak{E}\{T\}|T,\Theta) =& p(\mathfrak{O}\{\mathcal{L}(T)\}|\mathcal{L}(T),\Theta)p(\mathfrak{O}\{\mathcal{R}(T)\}|\mathcal{R}(T),\Theta) \\ &+p(\mathfrak{E}\{\mathcal{L}(T)\}|\mathcal{L}(T),\Theta)p(\mathfrak{E}\{\mathcal{R}(T)\}|\mathcal{R}(T),\Theta) \end{aligned}$$

   So, the recursions pertinent for getting an odd-sum output is to calculate both of:

   i the odd-sum output of left subtree and the even-sum output of right subtree;

   ii the even-sum output of left subtree and the odd-sum output of right subtree.

   And the recursions pertinent for getting an even-sum output is to calculate both of:

   i the odd-sum output of left subtree and the odd-sum output of right subtree;

   ii the even-sum output of left subtree and the even-sum output of right subtree.

   And for a single step, the marginal probability is calculated, such that

$$p(\mathfrak{O}\{T\}|T,\Theta) = \sum_{k=1}^{K} p(\mathfrak{O}\{T\}, X_{r(T)} = k|T,\Theta)$$

   where $X_{r(T)}$ means the root of the tree T.

2. **Top-down sampling algorithm:** To sample the leaves that with odd sum, we use Bayesian theorem so that (for simplicity, T and Θ are omitted)

$$p(X_{r(T)}|\mathfrak{O}\{T\}) = \frac{p(\mathfrak{O}\{T\}|X_{r(T)})p(X_{r(T)})}{p(\mathfrak{O}\{T\})}.$$

   Then, if we look at the left child of

Question 2.8.23: *Implement your bottom up DP algorithm for the probability of generating an odd sum output.*

```
import numpy as np

def tree_DP(root, k):
    # TODO: Implement algorithm for dynamic programming
    p = np.dot(root.cat, odd_DP(root, k))
```

```
    q = np.dot(root.cat, even_DP(root, k))
    print(p+q)
    return p

def odd_DP(node, k):
    if len(node.descendants) != 0:
        theta1 = np.array((node.descendants[0].cat))
        theta2 = np.array((node.descendants[1].cat))

        s11 = odd_DP(node.descendants[0], k)
        s21 = even_DP(node.descendants[1], k)
        s1  = np.dot(theta1, s11) * np.dot(theta2, s21)

        s12 = even_DP(node.descendants[0], k)
        s22 = odd_DP(node.descendants[1], k)
        s2  = np.dot(theta1, s12) * np.dot(theta2, s22)
        return s1 + s2

    s = np.zeros((k,1))
    s[1::2] = 1
    return s

def even_DP(node, k):
    if len(node.descendants) != 0:
        theta1 = np.array((node.descendants[0].cat))
        theta2 = np.array((node.descendants[1].cat))

        s11 = odd_DP(node.descendants[0], k)
        s21 = odd_DP(node.descendants[1], k)
        s1  = np.dot(theta1, s11) * np.dot(theta2, s21)

        s12 = even_DP(node.descendants[0], k)
        s22 = even_DP(node.descendants[1], k)
        s2  = np.dot(theta1, s12) * np.dot(theta2, s22)
        return s1 + s2

    s = np.zeros((k,1))
    s[0::2] = 1
    return s
```

Listing 4: Tree DP.

Question 2.8.24: *Implement your sampling algorithm.*

Question 2.8.25: *Apply your algorithm to the graphical model and data provided separately.*

9 Failing components VI

Question 2.9.26: *Derive a VI algorithm that estimates the posterior distribution for this model.*

From the question, we have

$$\begin{aligned} p(Z_1^n) =& \frac{1}{R} \\ p(Z_{c+1}^n | Z_c^n) =& 1 - \left(\frac{2}{3}\right)^{N_{Z_c^n}(Z_{c+1}^n)} \end{aligned}$$

where $c \leq 1$, and

$$N_{Z_c^n}(Z_{c+1}^n) = \begin{cases} 0, & \text{if } Z_{c+1}^n \notin N(Z_c^n) \\ 1, & \text{if } Z_{c+1}^n \in N(Z_c^n) \end{cases}$$

and define $N_{Z_0^n}(Z_1^n) = 0$. Also, we have

$$\begin{aligned} p(X_{r,c}^n | Z_c^n, \Theta_{r,c}) =& \text{Bernoulli}(X_{r,c}^n | \theta_f)^{1-I_r(Z_c^n)} \text{Bernoulli}(X_{r,c}^n | \theta_{r,c})^{I_r(Z_c^n)} \\ =& \theta_f^{X_{r,c}^n (1-I_r(Z_c^n))} (1-\theta_f)^{(1-X_{r,c}^n)(1-I_r(Z_c^n))} \\ & \theta_{r,c}^{X_{r,c}^n I_r(Z_c^n)} (1-\theta_{r,c})^{(1-X_{r,c}^n) I_r(Z_c^n)}, \end{aligned}$$

where

$$I_r(Z_c^n) = \begin{cases} 0, & \text{if } Z_c^n = r \\ 1, & \text{if } Z_c^n \neq r \end{cases}.$$

And also

$$\begin{aligned} p(\theta_f) p(X_{r,c} | Z_c = r, \Theta_{r,c}) =& \text{Bernoulli}(X_{r,c} | \theta_f) \text{Beta}(\theta_f | a_f, b_f) \\ =& \text{Beta}\left(\theta_f | a_f + \sum_{n=1}^N X_{r,c}^n (1 - I_r(Z_c^n)), b_f + \sum_{n=1}^N (1 - X_c^n)(1 - I_r(Z_c^n))\right) \\ p(\theta_{r,c}) p(X_{r,c} | Z_c \neq r, \Theta_{r,c}) =& \text{Bernoulli}(X_{r,c} | \theta_{r,c}) \text{Beta}(\theta_{r,c} | a, b) \\ =& \text{Beta}\left(\theta_{r,c} | a + \sum_{n=1}^N X_{r,c}^n I_r(Z_c^n), b + \sum_{n=1}^N (1 - X_{r,c}^n) I_r(Z_c^n)\right) \end{aligned}$$

from which we obtain

$$p(\Theta_{r,c}) p(X_{r,c} | Z_c, \Theta_{r,c}) = \text{Beta}(\theta_f | a'_f, b'_f) \text{Beta}(\theta_{r,c} | a', b')$$

where

$$\begin{aligned} a'_f =& a_f + \sum_{n=1}^N X_{r,c}^n (1 - I_r(Z_c^n)) & b'_f =& b_f + \sum_{n=1}^N (1 - X_c^n)(1 - I_r(Z_c^n)) \\ a' =& a + \sum_{n=1}^N X_{r,c}^n I_r(Z_c^n) & b' =& b + \sum_{n=1}^N (1 - X_{r,c}^n) I_r(Z_c^n) \end{aligned}$$

Also, since the model follows the same structure with Hidden Markov Model (HMM), then the joint distribution of all the random variables is given by

$$\begin{aligned}
p(X,Z,\Theta) =& p(Z_1)p(\Theta_1)P(X_1|Z_1,\Theta_1) \\
& p(Z_2|Z_1)p(\Theta_2)p(X_2|Z_2,\Theta_2) \\
& \cdots \\
& p(Z_C|Z_{C-1})p(\Theta_C)P(X_C|Z_C,\Theta_C) \\
=& p(Z_1)\prod_{r=1}^{R} p(\Theta_{r,1})P(X_{r,1}|Z_1,\Theta_{r,1}) \\
& p(Z_2|Z_1)\prod_{r=1}^{R} p(\Theta_{r,2})p(X_{r,2}|Z_2,\Theta_{r,2}) \\
& \cdots \\
& p(Z_C|Z_{C-1})\prod_{r=1}^{R} p(\Theta_{r,C})P(X_{r,C}|Z_C,\Theta_{r,C}) \\
=& p(Z)\prod_{c=1}^{C}\prod_{r=1}^{R} p(\Theta_{r,c})P(X_{r,c}|Z_c,\Theta_{r,c})
\end{aligned}$$

A variational distribution is considered so that

$$q(Z,\Theta) = q(Z)q(\Theta).$$

The log of the optimized factor is given by

$$\begin{aligned}
\ln q^*(Z) =& \ln q^*(Z_1) + \ln q^*(Z_2|Z_1) + \cdots + \ln q^*(Z_C|Z_{C-1}) \\
=& \mathbb{E}[\ln p(Z_1)] + \mathbb{E}[\ln p(Z_2|Z_1)] + \cdots + \mathbb{E}[\ln p(Z_C|Z_{C-1})] + \mathbb{E}_\Theta[p(X|Z,\Theta)] + \text{const} \\
=& \frac{N}{R} + \sum_{n=1}^{N}\sum_{c=1}^{C-1} \ln\left(1 - \left(\frac{2}{3}\right)^{N_{Z_c^n}(Z_{c+1}^n)}\right) + \text{const} \\
& + \sum_{n=1}^{N}\sum_{c=1}^{C}\sum_{r=1}^{R} \big\{X_{r,c}^n(1 - I_r(Z_c^n))\mathbb{E}[\ln\theta_f] + (1 - X_{r,c}^n)(1 - I_r(Z_c^n))\mathbb{E}[\ln(1-\theta_k)] \\
& + x_{r,c}^n I_r(Z_c^n)\mathbb{E}[\ln\theta_{r,c}] + (1 - X_{r,c}^n)I_r(Z_c^n)\mathbb{E}[\ln(1-\theta_{r,c})]\big\} \\
=& \sum_{n=1}^{N}\sum_{c=1}^{C}\sum_{r=1}^{R} \frac{1}{R}\ln\left(1 - \left(\frac{2}{3}\right)^{N_{Z_c^n}(Z_{c+1}^n)}\right) + \text{const} \\
& + \sum_{n=1}^{N}\sum_{c=1}^{C}\sum_{r=1}^{R} \big\{X_{r,c}^n(1 - I_r(Z_c^n))\mathbb{E}[\ln\theta_f] + (1 - X_{r,c}^n)(1 - I_r(Z_c^n))\mathbb{E}[\ln(1-\theta_k)] \\
& + x_{r,c}^n I_r(Z_c^n)\mathbb{E}[\ln\theta_{r,c}] + (1 - X_{r,c}^n)I_r(Z_c^n)\mathbb{E}[\ln(1-\theta_{r,c})]\big\} \\
=& \sum_{n=1}^{N}\sum_{r=1}^{R}\sum_{c=1}^{C} \ln\rho_{n,r,c} + \text{const}
\end{aligned}$$

where could have $\ln(0)$ problem, which can be solved in practice by calculating $\ln(sys.float\_info.epsilon)$

instead. Then take the exponential of both side of the equation and normalize it, we obtain

$$q^*(Z) = \prod_{n=1}^{N} \prod_{r=1}^{R} \prod_{c=1}^{C} r_{n,r,c}$$

where

$$r_{n,r,c} = \frac{\rho_{n,r,c}}{\sum_{r=1}^{R} \sum_{c=1}^{C} \rho_{n,r,c}}.$$

Then for $q(\Theta)$, we have

$$\ln q^*(\Theta_{r,c}) = \ln \left[\mathrm{Beta}(\theta_f | a'_f, b'_f)\right] + \ln \left[\mathrm{Beta}(\theta_{r,c} | a', b')\right].$$

So, we have

$$\begin{aligned} q^*(\Theta_{r,c}) =& q^*(\theta_f) q^*(\theta_{r,c}) \\ =& \mathrm{Beta}(\theta_f | a^*_f, b^*_f)\, \mathrm{Beta}(\theta_{r,c} | a^*_{r,c}, b^*_{r,c}) \end{aligned}$$

where

$$a^*_f = a'_f = a_f + \sum_{n=1}^{N} X^n_{r,c}(1 - I_r(Z^n_c)) \qquad b^*_f = b'_f = b_f + \sum_{n=1}^{N} (1 - X^n_c)(1 - I_r(Z^n_c))$$

$$a^* = a' = a + \sum_{n=1}^{N} X^n_{r,c} I_r(Z^n_c) \qquad b^* = b' = b + \sum_{n=1}^{N} (1 - X^n_{r,c}) I_r(Z^n_c)$$

References

[1] C. M. Bishop, *Pattern recognition and machine learning.* Information science and statistics, New York: Springer, 2006.

[2] J. Kruschke, *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan.* Academic Press, 2014.